

EL624352935US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
APPLICATION FOR LETTERS PATENT

**A SYSTEM AND METHOD
FOR
JOINT OPTIMIZATION OF LANGUAGE MODEL
PERFORMANCE AND SIZE**

DEPARTMENT OF COMMERCE

Inventor(s):

**Jianfeng Gao
Kai-Fu Lee
Mingjing Li
Haifeng Wang
Dongfeng Cai
Lee-Feng Chien**

ATTORNEY'S DOCKET NO. MS1-441US

This application claims priority to a provisional patent application No. 60/TBD, entitled "A Method for Iterative Joint Optimization of Language Model Perplexity and Size", filed on 11/5/99 by the inventors of this application.

TECHNICAL FIELD

This invention generally relates to language modeling and, more specifically, to a system and method for iterative joint optimization of language model performance and size.

BACKGROUND

Recent advances in computing power and related technology have fostered the development of a new generation of powerful software applications including web-browsers, word processing and speech recognition applications. The latest generation of web-browsers, for example, anticipate a uniform resource locator (URL) address entry after a few of the initial characters of the domain name have been entered. Word processors offer improved spelling and grammar checking capabilities, word prediction, and language conversion. Newer speech recognition applications similarly offer a wide variety of features with impressive recognition and prediction accuracy rates. In order to be useful to an end-user, these features must execute in substantially real-time. To provide this performance, many applications rely on a tree-like data structure to build a simple language model.

Simplistically, a language model measures the likelihood of any given sentence. That is, a language model can take any sequence of items (words, characters, letters, etc.) and estimate the probability of the sequence. The estimation performed by language models are typically made using a simple lexicon (e.g., a word list) and a segmentation algorithm or rules. A common

1 approach to building a prior art language model is to utilize a prefix tree-like data
2 structure to build an N-gram language model from a known training set of a textual
3 corpus.

4 The use of a prefix tree data structure (a.k.a. a suffix tree, or a PAT tree)
5 enables a higher-level application to quickly traverse the language model,
6 providing the substantially real-time performance characteristics described above.
7 Simplistically, the N-gram language model counts the number of occurrences of a
8 particular item (word, character, etc.) in a string (of size N) throughout a text. The
9 counts are used to calculate the probability of the use of the item strings.
10 Development of a typical tri-gram (N-gram where N=3) language model, for
11 example, generally includes the steps of:

- 12 (a) dissecting a received textual corpus into a plurality of items (characters,
13 letters, numbers, etc.);
- 14 (b) the items (e.g., characters (C)) are segmented (e.g., into words (W)) in
15 accordance with a small, pre-defined lexicon and a simple, pre-defined
16 segmentation algorithm, wherein each W is mapped in the tree to one or
17 more C's;
- 18 (c) train a language model on the dissected corpus by counting the
19 occurrence of strings of characters, from which the probability of a
20 sequence of words (W_1, W_2, \dots, W_M) is predicted from the previous two
21 words:

$$P(W_1, W_2, W_3, \dots, W_M) \approx \prod P(W_i \mid W_{i-1}, W_{i-2}) \quad (1)$$

25 The N-gram language model is limited in a number of respects. First, the
counting process utilized in constructing the prefix tree is very time consuming.

1 Thus, only small N-gram models (typically bi-gram, or tri-gram) can practically be
2 achieved. Second, as the string size (N) of the N-gram language model increases,
3 the memory required to store the prefix tree increases by 2^N . Thus, the memory
4 required to store the N-gram language model, and the access time required to
5 utilize a large N-gram language model is prohibitively large for N-grams larger
6 than three (i.e., a tri-gram).

7 As a consequence of these computational and architectural limitations, prior
8 art implementations of N-gram language models tend to be very rigid. That is,
9 prior art N-gram language models tend to use a standard (small) lexicon, a
10 simplistic segmentation algorithm, and will typically only rely on the previous two
11 words to predict the current word (in a tri-gram model).

12 A small lexicon limits the ability of the model to identify words to those
13 contained in the lexicon. If a word is not in the lexicon, it does not exist as far as
14 the model is concerned. Moreover, a basic multipurpose lexicon is not likely to
15 represent the linguistic complexity or syntactic behavior of a particular application
16 or style or writing. Thus, a small lexicon is not likely to cover the intended
17 linguistic content of a given application.

18 The segmentation algorithms are often ad-hoc and not based on any
19 statistical or semantic principles. A simplistic segmentation algorithm typically
20 errors in favor of larger words over smaller words. Thus, the model is unable to
21 accurately predict smaller words contained within larger, lexiconically acceptable
22 strings.

23 As a result of the foregoing limitations, a language model using prior art
24 lexicon and segmentation algorithms tend to be error prone. That is, any errors
25 made in the lexicon or segmentation stage are propagated throughout the language
model, thereby limiting its accuracy and predictive attributes.

In addition to the fundamental problems of a limited lexicon and a simplistic segmentation algorithm, the N-gram approach is fundamentally constrained by limiting the predictive features to at most the previous N-1 words. In the instance of a tri-gram (N=3) language mode (LM), the LM is limited to only the previous two words for context. These inherent limitations in the prior art of language modeling fundamentally constrain the accuracy of such language models.

In application, the prior art approach to language modeling may provide acceptable results in many alphabet-based languages with an accepted lexicon and well-defined segmentation. The aforementioned limitations inherent in such prior art language models are further exacerbated, however, when applied to numerical or character-based languages such as, for example, many Asian languages. The Chinese language, for example, is a character-based language with an expansive lexicon that is not well-defined, where single characters may form a word or may be combined with another character to form a multi-character word with a distinct and unique meaning in the language, and where there is limited punctuation to provide clues as to sentence structure, and the like. In such a language, lexicon and segmentation clues are difficult, at best, to come by. Prior art language modeling techniques provide very poor results when applied to such languages.

One proposed solution to improving the performance of a language model applied to character-based languages is to simply throw more data at the model, i.e., trade size for accuracy. The thought is that more data provides a larger lexicon and basis for maximum match-based segmentation algorithms (to be defined more fully below) to refine the language model. An obvious consequence to this solution however, and a significant limitation in and of itself, is that to simply throw more data at the model significantly increases the memory requirements required to support the language model. Aside from the cost of

1 providing the additional memory, larger language models place a greater
2 computational burden on the host system/application utilizing the language model.
3 The memory and computational consequences of the prior art solution typically
4 result in a modest improvement in predictive capability. Moreover, as above, a
5 huge data set does not necessarily provide an improved language model on a per-
6 application basis.

7 Thus, a system and method for the joint optimization of language model
8 performance and size is required, unencumbered by the deficiencies and
9 limitations commonly associated with prior art language modeling techniques.
10 Just such a solution is provided below.

11

12 **SUMMARY**

13 This invention concerns a system and method for iterative joint
14 optimization of language model performance and size. To overcome the
15 limitations commonly associated with the prior art, the present invention does not
16 rely on a predefined lexicon or segmentation algorithm, but rather dynamically
17 defines a lexicon and segmentation rules while iteratively refining a language
18 model. According to one implementation, a method for the joint optimization of
19 language model performance and size is presented comprising developing a
20 language model from a tuning set of information, segmenting at least a subset of a
21 received textual corpus and calculating a perplexity value for each segment and
22 iteratively refining the language model with one or more segments of the received
23 corpus based, at least in part, on the calculated perplexity value for the one or more
24 segments.

25 According to one implementation, the received corpus, or subset thereof, is
dissected into a plurality of training units, and a measure of similarity within, and

disparity between each training unit is made. Based on these measures, gaps for training chunks are selected which maximize the similarity within the chunks and the disparity between chunks. A perplexity value is calculated for each of the chunks, using the seed language model. In one embodiment, if the tuning set is not large enough to provide a robust seed language model, the tuning set is augmented with chunks having a measured perplexity value that exceeds some threshold. In this instance, the seed language model is then re-trained and the perplexity calculations are once again performed on a per-chunk basis and represent a set of data on which the language model will be trained (e.g., training data).

According to one aspect of the invention, the size of the training data may well be reduced by application of a perplexity value filter. The filter eliminates chunks from the training data with too large a perplexity value. The remaining training data is then combined to form the language model training data. According to one implementation of the invention, the training data may be combined by combining counts from each of the training chunks. The “count” of a chunk is a measure of the probability of a word combination weighted by the perplexity of the chunk. The probability of a word is measured using, e.g., a trigram language model, wherein the probability of a word is measured using the prior two words.

According to another embodiment, the training data is combined by combining models for each of the chunks. This approach involves clustering training chunks into a few clusters by quantization, training a language model on a per-cluster basis, and interpolating them, wherein the interpolation weights are estimated using an estimate maximize method.

A resultant language model is then compressed according to one of a number of alternate techniques. According to one implementation, the language

1 model compression is performed according to a relative entropy pruning algorithm.
2 Simply stated, the relative entropy pruning algorithm employed herein eliminates
3 as many chunks from the training set as necessary to satisfy memory and/or
4 application constraints, while minimizing the effect on the performance of the
5 resultant language model.

6

7 **BRIEF DESCRIPTION OF THE DRAWINGS**

8 The same reference numbers are used throughout the figures to reference
9 like components and features.

10 **Fig. 1** is a block diagram of a computer system incorporating the teachings
11 of the present invention;

12 **Fig. 2** is a block diagram of an example modeling agent incorporating the
13 teachings of the present invention;

14 **Fig. 3** is a graphical representation of a prefix tree language model data
15 structure according to one aspect of the present invention;

16 **Fig. 4** is a flow chart of an example method for building a prefix tree;

17 **Fig. 5** is a flow chart of an example method for the joint optimization of
18 language model performance and size, according to the teachings of the present
19 invention;

20 **Fig. 6** is a flow chart detailing an example method for automatically
21 segmenting a training set from a received corpus;

22 **Fig. 7** is a flow chart illustrating an example method for ranking the
23 segments of the training set in order of perplexity;

24 **Fig. 8** is a flow chart of an example method for combining the training data
25 to train a language model according to two alternate embodiments of the invention;
and

1 **Fig. 9** is a storage medium with a plurality of executable instructions which,
2 when executed, implement the innovative modeling agent of the present invention,
3 according to an alternate embodiment of the present invention.

4

5 **DETAILED DESCRIPTION**

6 This invention concerns a system and iterative method for the joint
7 optimization of language model performance and size. In the discussion herein,
8 the invention is described in the general context of computer-executable
9 instructions, such as program modules, being executed by one or more
10 conventional computers. Generally, program modules include routines, programs,
11 objects, components, data structures, etc. that perform particular tasks or
12 implement particular abstract data types. Moreover, those skilled in the art will
13 appreciate that the invention may be practiced with other computer system
14 configurations, including hand-held devices, personal digital assistants,
15 multiprocessor systems, microprocessor-based or programmable consumer
16 electronics, network PCs, minicomputers, mainframe computers, and the like. In a
17 distributed computer environment, program modules may be located in both local
18 and remote memory storage devices. It is noted, however, that modification to the
19 architecture and methods described herein may well be made without deviating
20 from spirit and scope of the present invention.

21

22 **EXAMPLE COMPUTER SYSTEM**

23 **Fig. 1** illustrates an example computer system 102 including an innovative
24 language modeling agent (LMA) 104, to jointly optimize the performance and size
25 of a language model, according to one embodiment of the present invention. As
used herein, language model performance is quantified in terms of a “perplexity

1 value". An equation for the perplexity value will be thoroughly developed below,
2 however, for ease of understanding the perplexity value may be thought of as a
3 language model predictive capability index. Simplistically speaking, the perplexity
4 value calculated for an item (character, letter, number, word, etc.) is inversely
5 related to the probability that the item appears in a given context. Thus, lower
6 perplexity values are better.

7 According to the teachings of the present invention, LMA 104 iteratively
8 refines the language model to improve performance, while simultaneously
9 reducing the size of the language model in accordance with system memory and/or
10 application constraints. It should be appreciated that although depicted as a
11 separate, stand alone application in Fig. 1, language modeling agent 104 may well
12 be implemented as a function of an application, e.g., word processor, web browser,
13 speech recognition system, etc. Moreover, although depicted as a software
14 application, those skilled in the art will appreciate that the innovative modeling
15 agent may well be implemented in hardware, e.g., a programmable logic array
16 (PLA), a special purpose processor, an application specific integrated circuit
17 (ASIC), microcontroller, and the like.

18 As used herein, computer 102 is intended to represent any of a class of
19 general or special purpose computing platforms which, when endowed with the
20 innovative language modeling agent (LMA) 104, implement the teachings of the
21 present invention in accordance with the first example implementation introduced
22 above. In this regard, but for the description of LMA 104, the following
23 description of computer system 102 is intended to be merely illustrative, as
24 computer systems of greater or lesser capability may well be substituted without
25 deviating from the spirit and scope of the present invention.

1 As shown, computer 102 includes one or more processors or processing
2 units 132, a system memory 134, and a bus 136 that couples various system
3 components including the system memory 134 to processors 132.

4 The bus 136 represents one or more of any of several types of bus
5 structures, including a memory bus or memory controller, a peripheral bus, an
6 accelerated graphics port, and a processor or local bus using any of a variety of bus
7 architectures. The system memory includes read only memory (ROM) 138 and
8 random access memory (RAM) 140. A basic input/output system (BIOS) 142,
9 containing the basic routines that help to transfer information between elements
10 within computer 102, such as during start-up, is stored in ROM 138. Computer
11 102 further includes a hard disk drive 144 for reading from and writing to a hard
12 disk, not shown, a magnetic disk drive 146 for reading from and writing to a
13 removable magnetic disk 148, and an optical disk drive 150 for reading from or
14 writing to a removable optical disk 152 such as a CD ROM, DVD ROM or other
15 such optical media. The hard disk drive 144, magnetic disk drive 146, and optical
16 disk drive 150 are connected to the bus 136 by a SCSI interface 154 or some other
17 suitable bus interface. The drives and their associated computer-readable media
18 provide nonvolatile storage of computer readable instructions, data structures,
19 program modules and other data for computer 102.

20 Although the exemplary environment described herein employs a hard disk
21 144, a removable magnetic disk 148 and a removable optical disk 152, it should be
22 appreciated by those skilled in the art that other types of computer readable media
23 which can store data that is accessible by a computer, such as magnetic cassettes,
24 flash memory cards, digital video disks, random access memories (RAMs) read
25 only memories (ROM), and the like, may also be used in the exemplary operating
environment.

A number of program modules may be stored on the hard disk 144, magnetic disk 148, optical disk 152, ROM 138, or RAM 140, including an operating system 158, one or more application programs 160 including, for example, the innovative LMA 104 incorporating the teachings of the present invention, other program modules 162, and program data 164 (e.g., resultant language model data structures, etc.). A user may enter commands and information into computer 102 through input devices such as keyboard 166 and pointing device 168. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit 132 through an interface 170 that is coupled to bus 136. A monitor 172 or other type of display device is also connected to the bus 136 via an interface, such as a video adapter 174. In addition to the monitor 172, personal computers often include other peripheral output devices (not shown) such as speakers and printers.

As shown, computer 102 operates in a networked environment using logical connections to one or more remote computers, such as a remote computer 176. The remote computer 176 may be another personal computer, a personal digital assistant, a server, a router or other network device, a network "thin-client" PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 102, although only a memory storage device 178 has been illustrated in Fig. 1.

As shown, the logical connections depicted in Fig. 1 include a local area network (LAN) 180 and a wide area network (WAN) 182. Such networking environments are commonplace in offices, enterprise-wide computer networks, Intranets, and the Internet. In one embodiment, remote computer 176 executes an Internet Web browser program such as the "Internet Explorer" Web browser

1 manufactured and distributed by Microsoft Corporation of Redmond, Washington
2 to access and utilize online services.

3 When used in a LAN networking environment, computer 102 is connected
4 to the local network 180 through a network interface or adapter 184. When used in
5 a WAN networking environment, computer 102 typically includes a modem 186 or
6 other means for establishing communications over the wide area network 182,
7 such as the Internet. The modem 186, which may be internal or external, is
8 connected to the bus 136 via a input/output (I/O) interface 156. In addition to
9 network connectivity, I/O interface 156 also supports one or more printers 188. In
10 a networked environment, program modules depicted relative to the personal
11 computer 102, or portions thereof, may be stored in the remote memory storage
12 device. It will be appreciated that the network connections shown are exemplary
13 and other means of establishing a communications link between the computers
14 may be used.

15 Generally, the data processors of computer 102 are programmed by means
16 of instructions stored at different times in the various computer-readable storage
17 media of the computer. Programs and operating systems are typically distributed,
18 for example, on floppy disks or CD-ROMs. From there, they are installed or
19 loaded into the secondary memory of a computer. At execution, they are loaded at
20 least partially into the computer's primary electronic memory. The invention
21 described herein includes these and other various types of computer-readable
22 storage media when such media contain instructions or programs for implementing
23 the innovative steps described below in conjunction with a microprocessor or other
24 data processor. The invention also includes the computer itself when programmed
25 according to the methods and techniques described below. Furthermore, certain
sub-components of the computer may be programmed to perform the functions and

1 steps described below. The invention includes such sub-components when they
2 are programmed as described. In addition, the invention described herein includes
3 data structures, described below, as embodied on various types of memory media.

4 For purposes of illustration, programs and other executable program
5 components such as the operating system are illustrated herein as discrete blocks,
6 although it is recognized that such programs and components reside at various
7 times in different storage components of the computer, and are executed by the
8 data processor(s) of the computer.

9

10 **Example Language Modeling Agent**

11 **Fig. 2** illustrates a block diagram of an example language modeling agent
12 (LMA) 104, incorporating the teachings of the present invention. As shown,
13 language modeling agent 104 is comprised of one or more controllers 202, an
14 innovative analysis engine 204, storage/memory device(s) 206 and, optionally, one
15 or more additional applications (e.g., graphical user interface, prediction
16 application, verification application, estimation application, language conversion
17 application, etc.) 208, each communicatively coupled as shown. It will be
18 appreciated that although depicted in Fig. 2 as a number of disparate blocks, one or
19 more of the functional elements of the LMA 104 may well be combined. In this
20 regard, modeling agents of greater or lesser complexity, which jointly optimize
21 language model perplexity and size may well be employed without deviating from
22 the spirit and scope of the present invention.

23 As alluded to above, although depicted as a separate functional element,
24 LMA 104 may well be implemented as a function of a higher-level application,
25 e.g., a word processor, web browser, speech recognition system, or a language
conversion system. In this regard, controller(s) 202 of LMA 104 are responsive to

one or more instructional commands from a parent application to selectively invoke the features of LMA 104. Alternatively, LMA 104 may well be implemented as a stand-alone language modeling tool, providing a user with a user interface (208) to selectively implement the features of LMA 104 discussed below. In either case, controller(s) 202 of LMA 104 selectively invoke one or more functions of analysis engine 204 (described more fully below) to jointly optimize language model performance and size. In this regard, controller 202 may receive a size constraint from a calling application, or independently identifies architectural memory constraints to determine the size of a generated language model.

As will be described more fully below, to jointly optimize language model performance and size, controller 202 iteratively refines a language model using a large textual corpus. The textual corpus is received and stored by controller 202 in storage device 208. Controller 202 begins the iterative optimization process with a “seed” language model developed from a tuning set of information, described more fully below. As above, the tuning set of information is received and stored by controller 202 in storage device 208.

Except as configured to effect the teachings of the present invention, controller 202 is intended to represent any of a number of alternate control systems known in the art including, but not limited to, a microprocessor, a programmable logic array (PLA), a micro-machine, an application specific integrated circuit (ASIC) and the like. In an alternate implementation, controller 202 is intended to represent a series of executable instructions to implement the control logic described above.

As shown, the innovative analysis engine 204 is comprised a Markov probability calculator 212, a frequency calculation function 210, a dynamic lexicon generation function 214, a dynamic segmentation function 216, and a data

1 structure memory manager 218. Upon receiving an external indication, controller
2 202 selectively invokes an instance of the analysis engine 204 to develop, modify
3 and optimize the performance and size of a statistical language model (SLM).

4 Analysis engine 204 invokes an instance of frequency analysis function 210
5 to identify the frequency and dependencies of items comprising at least a subset of
6 a received corpus (training set), to build a prefix tree data structure from the items
7 (e.g., letters, characters, numbers, etc.).

8 Frequency calculation function 213 identifies a frequency of occurrence for
9 each item (character, letter, number, word, etc.) in the training set subset. Based
10 on inter-node dependencies, data structure generator 210 assigns each item to an
11 appropriate node of the DOMM tree, with an indication of the frequency value (C_i)
12 and a compare bit (b_i). *(Handwritten notes: S, A, X, Y)*

13 The Markov probability calculator 212 calculates the probability of an item
14 (character, letter, number, etc.) from a context (j) of associated items. More
15 specifically, according to the teachings of the present invention, the Markov
16 probability of a particular item (C_i) is dependent on as many previous characters as
17 data “allows”, in other words:

$$19 P(C_1, C_2, C_3, \dots, C_N) \approx \prod P(C_i \mid C_{i-1}, C_{i-2}, C_{i-3}, \dots, C_j) \quad (2)$$

21 The number of characters employed as context (j) by Markov probability
22 calculator 212 is a “dynamic” quantity that is different for each sequence of
23 characters $C_i, C_{i-1}, C_{i-2}, C_{i-3}$, etc. According to one implementation, the number of
24 characters relied upon for context (j) by Markov probability calculator 212 is
25 dependent, at least in part, on a frequency value for each of the characters, i.e., the
rate at which they appear throughout the corpus. More specifically, if in

1 identifying the items of the corpus Markov probability calculator 212 does not
2 identify at least a minimum occurrence frequency for a particular item, it may be
3 “pruned” (i.e., removed) from the tree as being statistically irrelevant. According
4 to one embodiment, the minimum frequency threshold is three (3). Alternatively,
5 Markov probability calculator can also develop N-gram language models from a
6 training set.

7 As alluded to above, analysis engine 204 does not rely on a fixed lexicon or
8 a simple segmentation algorithm (both of which tend to be error prone). Rather,
9 analysis engine 204 selectively invokes a dynamic segmentation function 216 to
10 segment items (characters or letters, for example) into strings (e.g., words). More
11 precisely, segmentation function 216 segments the training set 222 into subsets
12 (chunks) and calculates a cohesion score (i.e., a measure of the similarity between
13 items within the subset). The segmentation and cohesion calculation is iteratively
14 performed by segmentation function 216 until the cohesion score for each subset
15 reaches a predetermined threshold.

16 The lexicon generation function 214 is invoked to dynamically generate and
17 maintain a lexicon 220 in memory 206. According to one implementation, lexicon
18 generation function 214 analyzes the segmentation results and generates a lexicon
19 from item strings with a Markov transition probability that exceeds a threshold. In
20 this regard, lexicon generation function 214 develops a dynamic lexicon 220 from
21 item strings which exceed a pre-determined Markov transition probability taken
22 from one or more language models developed by analysis engine 204.
23 Accordingly, unlike prior art language models which rely on a known, fixed
24 lexicon that is prone to error, analysis engine 204 dynamically generates a lexicon
25 of statistically significant, statistically accurate item strings from one or more
language models developed over a period of time. According to one embodiment,

1 the lexicon 220 comprises a “virtual corpus” that Markov probability calculator
2 212 relies on (in addition to the dynamic training set) in developing subsequent
3 language models.

4 Analysis engine 204 accesses information in storage device 208 by invoking
5 an instance data structure memory manager 218. According to one aspect of the
6 invention, data structure memory manager 218 utilizes system memory as well as
7 extended memory to maintain the resultant language model data structure.
8

9 **Example Data Structure – Dynamic Order Markov Model (DOMM) Tree**

10 Fig. 3 graphically represents a conceptual illustration of an example
11 Dynamic Order Markov Model tree-like data structure 300, according to the
12 teachings of the present invention. To conceptually illustrate how a DOMM tree
13 data structure 300 is configured, Fig. 3 presents an example DOMM data structure
14 300 for a language model developed from the English alphabet, i.e., A, B, C, ...Z.
15 As shown the DOMM tree 300 is comprised of one or more root nodes 302 and
16 one or more subordinate nodes 304, each associated with an item (character, letter,
17 number, word, etc.) of a textual corpus, logically coupled to denote dependencies
18 between nodes. According to one implementation of the present invention, root
19 nodes 302 are comprised of an item and a frequency value (e.g., a count of how
20 many times the item occurs in the corpus). At some level below the root node
21 level 302, the subordinate nodes are arranged in binary sub-trees, wherein each
22 node includes a compare bit (b_i), an item with which the node is associated
23 (A, B, ...), and a frequency value (C_N) for the item.

24 Thus, beginning with the root node associated with the item B 306, a binary
25 sub-tree is comprised of subordinate nodes 308-318 denoting the relationships
between nodes and the frequency with which they occur. Given this conceptual

example, it should be appreciated that starting at a root node, e.g., 306, the complexity of a search of the DOMM tree approximates $\log(N)$, where N is the total number of nodes to be searched.

As alluded to above, the size of the DOMM tree 300 may exceed the space available in the memory device 206 of LMA 104 and/or the main memory 140 of computer system 102. Accordingly, data structure memory manager 218 facilitates storage of a DOMM tree data structure 300 across main memory (e.g., 140 and/or 206) into an extended memory space, e.g., disk files on a mass storage device such as hard drive 144 of computer system 102.

Example Operation and Implementation

Having introduced the functional and conceptual elements of the present invention with reference to Figs. 1-3, the operation of the innovative language modeling agent 104 will now be described with reference to Figs. 4-9.

Building DOMM Tree Data Structure

Fig. 4 is a flow chart of an example method for building a Dynamic Order Markov Model (DOMM) data structure, according to one aspect of the present invention. As alluded to above, language modeling agent 104 may be invoked directly by a user or a higher-level application. In response, controller 202 of LMA 104 selectively invokes an instance of analysis engine 204, and a textual corpus (or a subset thereof) is loaded into memory 206 as a dynamic training set 222 and split into subsets (e.g., sentences, lines, etc.), block 402. In response, data structure generator 210 assigns each item of the subset to a node in data structure and calculates a frequency value for the item, block 404. According to one implementation, once data structure generator has populated the data structure with

1 the subset, frequency calculation function 213 is invoked to identify the occurrence
2 frequency of each item within the training set subset.

3 In block 406, data structure generator 210 determines whether additional
4 subsets of the training set remain and, if so, the next subset is read in block 408
5 and the process continues with block 404. In alternate implementation, data
6 structure generator 210 completely populates the data structure, a subset at a time,
7 before invocation of the frequency calculation function 213. In alternate
8 embodiment, frequency calculation function 213 simply counts each item as it is
9 placed into associated nodes of the data structure.

10 If, in block 406 data structure generator 210 has completely loaded the data
11 structure 300 with items of the training set 222, data structure generator 210 may
12 optionally prune the data structure, block 410. A number of mechanisms may be
13 employed to prune the resultant data structure 300.

14

15 **Example Method for Joint Optimization of Language Model Performance and Size**

16 **Fig. 5** is a flow chart of an example method for joint optimization of
17 language model performance and size, according to one embodiment of the present
18 invention. As shown, the method begins with block 400 wherein LM 104 is
19 invoked and a prefix tree of at least a subset of the received corpus is built. More
20 specifically, as detailed in Fig. 4, data structure generator 210 of modeling agent
21 104 analyzes the received corpus and selects at least a subset as a training set, from
22 which a DOMM tree is built.

23 In block 502, an initial language model is built from a tuning set of data
24 received or selected by controller 202. According to one embodiment, when
25 invoked by a higher level application, the application may provide LMA 104 with
a tuning set of data from which to generate a language model. Preferably, the

tuning set is comprised of one or more documents of scrubbed, application specific data representative of the linguistic and syntactic complexity for which the language model is intended. In this regard, the tuning set is preferably provided by a calling application. Alternatively, controller 202 may generate its own tuning set from stored documents, generate a tuning set from a prior language model, or utilize a default tuning set. The language model generated from the tuning set is referred to as the seed language model, or seed LM.

According to one implementation, controller 202 invokes an instance of Markov probability calculator (described above) to generate the seed LM from the tuning set. Alternatively, controller 202 may employ any of a number of prior art language modeling techniques such as, for example, a tri-gram language modeling technique. In either case, the seed language model will be refined with information from the received corpus to improve performance of the language model, while compressing the language model according to memory and/or application constraints.

Once the seed LM is developed, LMA 104 automatically segments a training set from the received corpus into a number (N) of chunks, block 504. According to one implementation, to be described more fully below, controller 202 invokes an instance of dynamic segmentation function 216 to automatically segment the corpus a number of chunks (N) satisfying a size range constraint (e.g., 500 items), maximizing the similarity within chunks and the disparity between chunks.

Once segmented, controller 202 ranks the chunks of the training set in order of increasing perplexity between the chunks, block 506. According to one implementation, controller 202 calculates a perplexity value for each of the chunks, wherein the perplexity value represents the prediction power of the current

iteration of the language model to the training chunk. More particularly, the perplexity value quantifies the possible number of words following a word in the training chunk on average based on the prediction of the current iteration of the language model. According to one implementation, controller 202 utilizes the following perplexity value equation:

$$PP = 2^{(\frac{1}{N}) \sum_{i=1}^N \log P(w_i|w_{i-1})} \quad (4)$$

As shown above, N represents the length (number of items) in the training chunk; P represents the probability of a word (w_i) given an immediately preceding word (w_{i-1}), e.g., a bi-gram language model. Thus, the perplexity value represents the prediction power with the certain language model to the training chunk. Starting with the seed LM, the complexity of word frequencies and word associations conceived in the application providing the tuning set can be modeled. In other words, the present invention provides for custom language models particularly relevant for poetry, scientific descriptions, mathematical proofs, Chinese language modeling, etc.

Once the training set has been ranked, the training data is combined to develop a trigram backoff language model, block 508. In this regard, controller 202 may combine the training data in any of a number of ways. For purposes of illustration, controller 202 may well combine the training chunks by combining the “counts” (or probabilities) of different chunk sets weighted by a measure of similarity within the chunk set. Alternatively, controller 202 may build a distinct LM for each distinct chunk and, utilizing an optimized interpolation algorithm, combine the models of the individual language models. Both will be described in greater detail below, with reference to Fig. 8.

Once the training data is combined, LMA 104 performs language model compression based, at least in part, on memory and/or application constraints. More particularly, controller 202 has developed an initial language model by combining the training data, the language model is iteratively refined to accommodate the size constraints while minimizing any adverse affect on language model performance, block 510. According to one aspect of the present invention, controller 202 utilizes a relative entropy-based pruning algorithm. Conceptually, controller 202 removes as many un(der)-utilized probabilities as possible without increasing the language model perplexity. In this regard, controller 202 examines the weighted relative entropy between each probability $P(w|h)$ and its value $P'(w|h')$ from the backoff distribution. Mathematically, the this distance is expressed as follows:

$$D(P(w|h), P'(w|h')) = P(w|h) * \log(P(w|h)/P'(w|h')) \quad (5)$$

As used herein, h is a history and h' is a reduced history. For small distances the backoff probability itself is good approximation and $P(w|h)$ does not carry much additional information. In such a case, controller 202 deletes this probability from the model. The deleted probability mass is reassigned to the backoff distribution, and controller 202 recalculates the backoff weights.

Having introduced the innovative operation of language modeling agent 104, more detailed flow charts are presented in Figs. 6-8 detailing aspects of the invention.

Fig. 6 illustrates a flow chart of an example method for automatically segmenting at least a subset of the received corpus, according to one implementation of the invention. As shown, the method begins in block 602 wherein dynamic segmentation function 216 empirically clusters every N items

1 into a *training unit*. According to one implementation, a training unit is comprised
2 of 500 non-stop items.

3 In block 604, controller 202 measures the statistical similarity within a
4 sequence of training units, referred to as a *training block*, on each side of a gap
5 separating training units. The measure of similarity between training blocks on
6 each side of a gap is referred to as a Cohesion score. According to one
7 implementation, the cohesion score is estimated by computing a correlation
8 coefficient of term vectors of the training blocks. One measure of such term
9 vectors is the well-known Term Frequency Inverse Document Frequency (TFIDF).
10 According to the illustrated example implementation, however, the term vectors
11 are estimated using only the term frequency, without taking into account the
12 inverse document frequency. In this regard, controller 202 invokes an instance of
13 frequency analysis function 213 to identify term frequency rates.

14 In block 606, controller 202 measures the statistical disparity between
15 training blocks on both sides of a gap between each training unit. The measure of
16 disparity, or the difference between the training blocks, is referred to as a Depth
17 score. The depth score at a gap can be estimated by calculating the difference in
18 cohesion scores on either side of the gap. Mathematically, the depth score may be
19 represented as:

$$D = (s_{i-1} - s_i) + (s_{i+1} - s_i) \quad (6)$$

21 In the equation above, s_i is the cohesion score at gap i .

22 In block 608, segmentation function 216 selects training chunk boundaries
23 at gaps wherein the depth score reaches a dynamically defined threshold.
24 According to one implementation, the threshold is dynamically defined as a
25 function of the mean (μ) and variance (σ) of the computed depth scores. In this

1 implementation, a gap is selected wherein the depth score of the gap is higher than
2 $\mu\text{-}c\sigma$, where c is a weighting factor (e.g., .5).

3 Once the training chunk boundaries are selected, block 608, the training set
4 is further refined by pruning the training chunks according to a size range
5 constraint, block 610. According to one implementation, segmentation function
6 216 utilizes a size range constraint to avoid too small or too large training chunks
7 in the resultant training set. In one implementation, segmentation function 216
8 utilizes a size range constraint to ensure training chunks of an average 10kB size

9 Once the training set has been segmented into training chunks (Fig. 6), the
10 training chunks are ranked according to a perplexity score. **Fig. 7** illustrates a flow
11 chart of an example method for ranking training chunks according to the teachings
12 of the present invention. As shown, the method begins with block 702 wherein
13 controller 202 calculates a perplexity score for each training chunk representing
14 the predictive power of the current LM to the training chunk. According to one
15 implementation, introduced above, controller 202 utilizes the perplexity
16 calculation (equation 5) to calculate a perplexity value for each of the training
17 chunks of the refined training set.

18 In block 704, controller 202 sorts the training chunks according to their
19 measured perplexity. According to one implementation, controller 202 sorts the
20 training chunks in order of increasing perplexity.

21 In block 706, a controller 202 determines whether the tuning set 220 is large
22 enough to satisfy a size constraint. If the tuning set is not large enough, there is a
23 risk that the resultant seed language model is impaired, which is then propagated to
24 the resultant language model through flawed perplexity calculations and training
25 chunk rankings. To alleviate this problem, controller 202 utilizes a dynamic size
constraint, e.g., a function of the size of the received corpus, to test the size of the

1 tuning set. If the tuning set is large enough to satisfy this size constraint, the
2 process continues with combining of the training chunks, block 508 (detailed in
3 Fig. 8).

4 If, however, the tuning set does not satisfy the size constraint, controller
5 202 augments the tuning set with the top N training chunks, block 708. That is,
6 controller 202 identifies the top N chunks with the lowest measured perplexity, and
7 adds the items represented by these chunks to the tuning set. In this way, the
8 tuning set is increased to meet the size range constraints, and is augmented with
9 the best available data. Once the tuning set has been augmented, the process of
10 blocks 502 through 506 (Fig. 5) must be repeated, albeit with the improved tuning
11 set.

12 **Fig. 8** illustrates a flow chart of an example method for combining the
13 resultant training chunks for language model training, block 508. As shown, the
14 method begins with block 802, wherein controller 202 filters the ranked training
15 chunks based, at least in part, on the received memory and/or application
16 constraints. Unlike prior art methods of filtering, however, controller 202 does not
17 simply remove the chunks with a poor perplexity measure. Rather, controller 202
18 keeps a certain number of similar chunks with relatively low perplexity and throws
19 out “bad” chunks, e.g., those with a perplexity value exceeding some threshold.

20 In block 804, the remaining training chunks are combined using one of two
21 alternate methods, colloquially referred to as (1) combining counts, or (2)
22 combining models. To represent the divergent methods, the flow chart of Fig. 8
23 illustrates a dashed line departing block 804 to block 902 or block 1002. Each of
24 the divergent paths represent alternate methods for combining the training chunks.
25

Combining counts

1 The combining counts method begins at block 902, wherein controller 202
2 determines the count (frequency) for each of the disparate training chunks.
3 According to one implementation, controller 202 calculates the count using the
4 equation:

5

$$6 \quad C(w_i, w_{i-1}, w_{i-2}) = \lambda_j C^j(w_i, w_{i-1}, w_{i-2}) \quad (7)$$

7

8 In the equation above, $C^j(w_i, w_{i-1}, w_{i-2})$ is the count of the trigram within training
9 chunk j , and is weighted with a measure of the training chunk perplexity (λ_j). The
10 weighting measure λ_j is defined by the following equation:

11

$$12 \quad \lambda_j = (1/PP_j)/(1/ PP_o) \quad (8)$$

13

14 That is, the weighting value is the ratio of the perplexity of the training chunk j to
15 the perplexity of the tuning set.

16 Once the count for each of the training chunks is determined, the counts
17 from the training chunks are combined, weighted by their occurrence frequency
18 throughout the training set, block 904.

19

20 Combining Models

21 An alternative approach to combining the training chunks is the combining
22 models approach. As shown, the process begins with block 1002 wherein
23 controller 202 clusters the training chunks into a few clusters. According to one
24 embodiment, controller 202 clusters the training chunks into a few clusters by
25 quantization, e.g., clustering training units with a similar perplexity value.

Once clustered, controller trains a language model for each of the clusters.
According to one implementation, controller 202 invokes an instance of Markov probability calculator to generate the cluster language models. In alternate embodiments, N-gram language models are utilized by controller 202.

In either case, the resultant cluster language models are interpolated, wherein the interpolation weights are estimated using an estimated maximize algorithm (also known as a forward-backward algorithm) which is an iterative procedure (or a gradient descent technique) for parameter optimization.

According to one embodiment, the resultant cluster language models are merged to form a composite language model "mixture". Initial experimental results indicate that such a merging may be performed without a degradation in LM performance, and offers computational decoding advantages. According to one embodiment, the language model is merged using linear interpolation. That is, the probability of a word, w, is the linear interpolation of 2 merge LMs. I.e. $P(w)=P1(w)+\alpha*P2(w)$, where $P1(w)$ is the probability from LM1, $P2(w)$ is from LM2, α is the interpolation weight, which is estimated by using EM algorithm.

Alternate Embodiments

Fig. 9 is a block diagram of a storage medium having stored thereon a plurality of instructions including instructions to implement the innovative modeling agent of the present invention, according to yet another embodiment of the present invention. In general, Fig. 9 illustrates a storage medium/device 900 having stored thereon a plurality of executable instructions 950 including at least a subset of which that, when executed, implement the innovative modeling agent 104 of the present invention. When executed by a processor of a host system, the

1 executable instructions 950 implement the modeling agent to generate a statistical
2 language model representation of a textual corpus for use by any of a host of other
3 applications executing on or otherwise available to the host system.

4 As used herein, storage medium 900 is intended to represent any of a
5 number of storage devices and/or storage media known to those skilled in the art
6 such as, for example, volatile memory devices, non-volatile memory devices,
7 magnetic storage media, optical storage media, and the like. Similarly, the
8 executable instructions are intended to reflect any of a number of software
9 languages known in the art such as, for example, C++, Visual Basic, Hypertext
10 Markup Language (HTML), Java, eXtensible Markup Language (XML), and the
11 like. Moreover, it is to be appreciated that the storage medium/device 900 need
12 not be co-located with any host system. That is, storage medium/device 900 may
13 well reside within a remote server communicatively coupled to and accessible by
14 an executing system. Accordingly, the software implementation of Fig. 9 is to be
15 regarded as illustrative, as alternate storage media and software embodiments are
16 anticipated within the spirit and scope of the present invention.

17 Although the invention has been described in language specific to structural
18 features and/or methodological steps, it is to be understood that the invention
19 defined in the appended claims is not necessarily limited to the specific features or
20 steps described. Rather, the specific features and steps are disclosed as exemplary
21 forms of implementing the claimed invention.